(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/072145 A1

(51) International Patent Classification⁷: G06F 11/00

(21) International Application Number: PCT/US00/14248

(22) International Filing Date: 24 May 2000 (24.05.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/318,204 25 May 1999 (25.05.1999) US

(71) Applicant: **TERADYNE, INC.** [US/US]; 321 Harrison Avenue, Boston, MA 02118 (US).

(72) Inventors: **APFELBAUM, Larry**; 89 Chestnut Street, West Newton, MA 02465 (US). **SAVAGE, Peter, L.**; 6 Westgate Road, Mont Vernon, NH 03057 (US). **BELL, Katrin**; 144 Eastern Avenue #203, Manchester, NH 03104 (US).

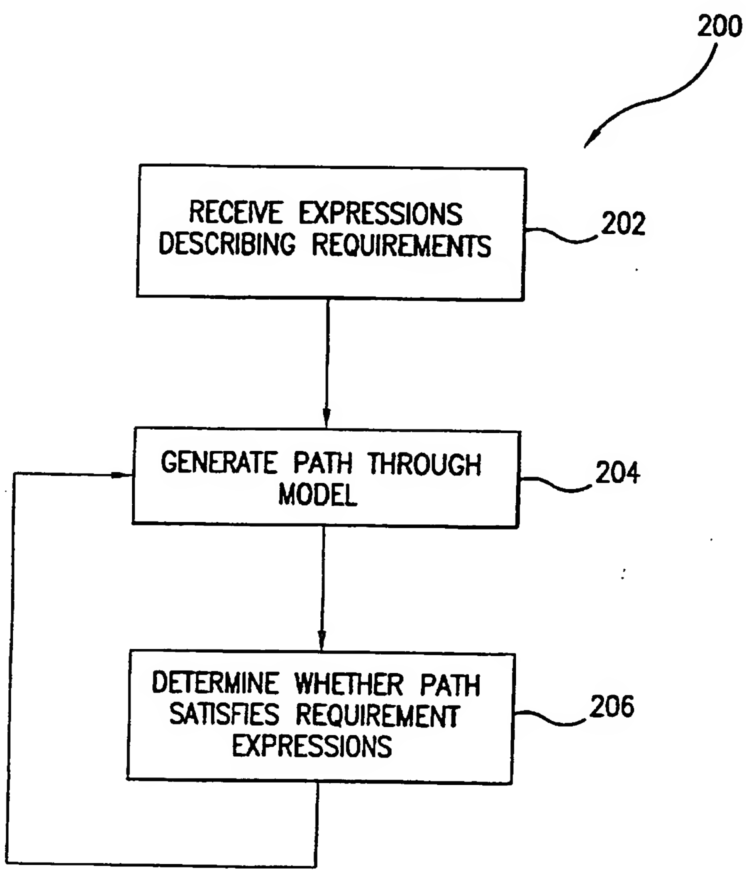
(74) Agent: **WALSH, Edmund, J.**; Teradyne, Inc., 321 Harrison Avenue, Boston, MA 02118 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

[Continued on next page]

(54) Title: ANALYZING AN EXTENDED FINITE STATE MACHINE SYSTEM MODEL



(57) Abstract: A method of using a computer to analyze an extended finite state machine model of a system includes receiving at least one requirement expression, determining at least one path of states and transitions through the model, evaluating at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression, and generating a report based on the evaluating.

WO 00/072145 A1

patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(48) Date of publication of this corrected version:
4 July 2002

(15) Information about Correction:
see PCT Gazette No. 27/2002 of 4 July 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Analyzing an Extended Finite State Machine System Model

Background of the Invention

System testing contributes significantly to system development and maintenance costs. TestMaster® software sold by Teradyne® Software and System Test, Inc. of Nashua, NH can reduce testing costs while increasing testing quality.

Referring to FIG. 1, TestMaster® software 100 enables a designer to create 102 an extended finite state machine model of a system. An extended finite state machine is represented by a directed graph that includes states interconnected by transitions. The software 100 provides a graphical user interface that enables the designer to "draw" the model by defining the states and connecting them together with directional lines that represent transitions. The model is independent of the system being modeled and can be created before or after the system is developed.

After the designer creates 102 the model, the software 100 detects 104 paths through the model states and transitions and generates 106 testing programs corresponding to each of the detected paths. Execution of the generated testing programs can identify system design flaws and highlight differences between the model created and the actual behavior of the underlying system.

Referring to FIG. 2, an extended finite state machine model 108 of a system includes states 110-116 interconnected by transitions 118-124. For example, as shown, a model 108 includes states 110-116 and transitions

118-124 representing a bank machine system that dispenses cash to customers entering an authorized PIN (Personal Identification Number).

The TestMaster® system automatically detects different paths through the model 108. For example, as shown in FIG. 3, a path through the model can include model elements A - T_{AB} - B - T_{BC} - C - T_{CD} - D. This path corresponds to a customer correctly entering an authorized PIN and successfully withdrawing cash. As shown in FIG. 4, a different path through the model can include model elements A - T_{AB} - B - T_{BD} - D. This model path corresponds to a customer who fails to correctly enter an authorized PIN.

TestMaster® offers many different procedures for detecting paths through a model. For example, a user can select from comprehensive, transition-based, N-switch, and quick-cover path detection. Comprehensive path detection outputs a test for every possible path through the model. Transition based path detection outputs tests such that each transition is included in at least one test. N-switch path detection outputs tests such that each unique sequence of N+1 transitions are included in at least one test. Comprehensive, transition, and N-switch path detection are currently implemented using a depth-first search. In contrast, quick-cover uses a "top-down" search and can output tests such that no transition is used more than a specified number of times. U.S. Patent Serial No. 08/658,344 entitled "Method and Apparatus for Adaptive Coverage in Test Generation" describes implementations of

programs for detecting extended finite state machine paths.

Referring again to FIG. 2, in addition to transitions and states, a model can incorporate variables and expressions that further define the model's behavior. TestMaster® can evaluate the expressions to assign variable values (e.g., y = mx + b) or to determine whether an expression is TRUE or FALSE (e.g., A AND (B OR C)). The expressions can include operators, variables, and other elements such as the names of states, transitions, and/or sub-models. When a named state, transition, or sub-model is in included in an expression, the model element evaluates to TRUE when included in the path currently being detected. For example, in FIG. 2, an expression of "(A && B)" would evaluate to TRUE for path portion "A - T_{AB} - B". As shown, expressions can use a PFL (Path Flow Language) syntax that resembles the C programming language. PFL and functions that can be called from PFL are described in The TestMaster® Reference Guide published by Teradyne®.

A model designer can associate the expressions with model elements to further define model behavior. For example, a designer can associate predicates and/or constraints with different states, transitions, and/or sub-models. Both predicates and constraints are evaluated during path detection and determine which transitions can be included in a path.

When path detection instructions encounter a model element having an associated predicate, the predicate expression is evaluated. If the predicate evaluates to TRUE, the model element associated with the predicate can be

used in the path. For example, as shown in FIG. 2, transition T_{BD} 124 has an associated predicate 126 ("!OKPin") that determines when a path can include the transition. As shown, the predicate 126 is a boolean expression that permits inclusion of the transition 124 in a path being detected when the boolean variable OKPin is FALSE and the path being detected has reached state B.

Similarly, when path detection instructions encounter a model element having an associated constraint, the constraint expression is evaluated. If the constraint evaluates to FALSE, the model element associated with the constraint cannot be used in the path being detected. For example, as shown in FIG. 2, a transition 123 can connect a state 114 to itself. To prevent a path from including a large or possibly infinite number of the same transition in a single path, a designer can specify a constraint expression 125 that limits use of a transition in a path. The "Iterate(3)" expression associated with the transition 123 limits a path through the model to including transition 123 three times. Thus, if evaluated at state C after looping around transition T_{CC} three times, the constraint would evaluate to FALSE and prevent further use of the transition in the current path. The constraint acts as a filter, eliminating generation of unwanted testing programs.

Referring to FIG. 5, a model can also include one or more sub-models. For example, the box labeled "EnterPIN" in FIG. 2 may be a sub-model 112 that includes additional states 128-136, transitions 138-150, and expressions. As shown, the sub-model 112 sets 150 the model variable OKPin

to TRUE when the customer PIN equals 1 148; otherwise, the sub-model sets the model variable OKPin to FALSE 146.

Sub-models encourage modular system design and increase comprehension of a model's design. Referring to FIG. 6, when the software 100 detects different paths through the system, the sub-model is essentially replaced with the states and transitions included in the sub-model.

Referring again to FIG. 5, a designer can define more than one transition 138-142 between states 128, 130. The designer can also associate expressions (e.g., PIN = 1) with each transition 138-142, for example, to set model variables to different values. For example, as shown, a designer has defined three transitions between the "Entry" 128 and "PINEntry" 130 states that each set a PIN variable to different value. Defining multiple transitions between states increases the number of paths through a model. For example, paths through the sub-model 112 can include I - $T_{IJ(1)}$ - J - T_{JK} - K - T_{KM} - M, I - $T_{IJ(2)}$ - J - T_{JL} - L - T_{LM} - M, and I - $T_{IJ(3)}$ - J - T_{JL} - L - T_{LM} - M. The use of multiple transitions enables testing of different conditions within the same model.

Summary of the Invention

In general, in one aspect, a method of using a computer to analyze an extended finite state machine model of a system includes receiving at least one requirement expression, determining at least one path of states and transitions through the model, evaluating at least one of the requirement expressions based on at least one of the

determined paths through the model to determine whether the path satisfies the requirement expression, and generating a report based on the evaluating.

Embodiments may include one or more of the following. The report may include a report of paths satisfying at least one requirement. The report may include a report of requirements satisfied by a path. The requirement expression may be a boolean expression. The requirement expression may include a variable, an operator, a state, a transition, a sub-model, a table-model, and/or another requirement expression. The method may generate testing programs only for paths satisfying a specified requirement or set of requirements.

In general, in another aspect, a method of using a computer to analyze an extended finite state machine model of a system includes receiving a requirement expression, determining at least one path of states and transitions through the model, evaluating at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression, and including the requirement expression as an element in a different expression, the evaluation of the requirement expression in the different expression depending on the determination of whether the path satisfies the requirement expression.

In general, in another aspect, a method of using a computer to analyze an extended finite state machine model of a system includes receiving at least one assertion expression, evaluating the assertion expression based on a

path being determined through the model, and if the evaluation indicates the path being determined fails to satisfy the assertion expression, indicating the failure.

Embodiments may include one or more of the following features. The method may further include halting model path determination. The assertion expression may be a boolean expression. The assertion expression may include a variable, an operator, a state, a transition, a sub-model, a table-model, and/or a requirement. The method may further include initiating a debugger if the evaluation indicates the path being determined fails to satisfy the assertion expression. The method may include receiving user input specifying when the assertion expression is evaluated. Specifying when the assertion expression is evaluated may be performed by specifying a model element.

The method may include automatically evaluating the assertion expression after a complete path through the model has been determined and/or automatically evaluating the assertion expression as model elements are added to a path being determined.

In general, in another aspect, a computer program product, disposed on a computer readable medium, for analyzing an extended finite state machine model of a system, includes instructions for causing a processor to receive at least one requirement expression, determine at least one path of states and transitions through the model, evaluate at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement

expression, and generate a report based on the evaluating.

In general, in another aspect, a computer program product, disposed on a computer readable medium, for analyzing an extended finite state machine model of a system includes instructions for causing a processor to receive a requirement expression, determine at least one path of states and transitions through the model, evaluate at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression, and include the requirement expression as an element in a different expression, the evaluation of the requirement expression in the different expression depending on the determination of whether the path satisfies the requirement expression.

In general, in another aspect, a computer program product, disposed on a computer readable medium, for analyzing an extended finite state machine model of a system includes instructions for causing a processor to receive at least one assertion expression, evaluate the assertion expression based on a path being determined through the model, and if the evaluation indicates the path being determined fails to satisfy the assertion expression, indicate the failure.

Brief Description of the Drawings

These and other features of the invention will become more readily apparent from the following detailed description when read together with the accompanying drawings, in which:

FIG. 1 is a flowchart of a process for using an extended finite state machine model to generate tests for a system according to the PRIOR ART;

FIG. 2 is a diagram of an extended finite state machine model according to the PRIOR ART;

FIGS. 3 and 4 are diagrams of paths through the extended finite state machine model of FIG. 2 according to the PRIOR ART;

FIG. 5 is a diagram of a sub-model according to the PRIOR ART;

FIG. 6 is a diagram of the extended finite state machine model that includes the states and transitions of the sub-model of FIG. 5 according to the PRIOR ART;

FIG. 7 is a flowchart of a process for determining whether a system model satisfies system requirements;

FIG. 8 is a screenshot of a table of system requirements used by the process of FIG. 7;

FIG. 9 is a screenshot of a requirements report produced by the process of FIG. 7;

FIG. 10 is a flowchart of a process for determining whether a system model satisfies specified assertions;

FIG. 11 is a diagram of an extended finite state machine model that includes a table model element;

FIG. 12 is a diagram of a table having rows incorporated into the model;

FIG. 13 is a flowchart of a process for selecting a transition based on likelihood values associated with the transitions;

FIG. 14 is a flowchart of a process for importing

data and other information into an extended finite state machine model;

FIG. 15 is a listing of a comma separated value file having values that can be imported into an extended finite state machine table model element;

FIG. 16 is a flowchart of a process for detecting paths through a model that conform to a user specified mix of paths; and

FIG. 17 is a diagram of a finite state machine model that includes model elements having target mix values.

Description of the Preferred Embodiments

Introduction

The inventors have invented different mechanisms that enable testers, developers, and others to detect design and implementation flaws in a system. These mechanisms can be included in TestMaster® or other software or hardware systems.

Requirements and Assertions:

Referring to FIG. 7, prose descriptions of system requirements often appear in functional and design specifications or are included in requirement documents produced by a customer. Requirements can also be gleaned from customers, bug-lists, etc. As shown in FIG. 7, a process 200 enables users to specify 202 requirements as an expression of elements (e.g., variables, sub-models, states, and transitions). For each path 204 through a model, the

process 200 evaluates 206 all requirement expressions to determine which requirements are satisfied.

For example, referring again to FIG. 2, the bank machine system functional specification may describe a requirement that no withdrawals should occur if a customer's PIN is not authorized. A user can ensure compliance with this requirement by defining a boolean expression of "NOT (withdrawal AND (NOT OKPin))". After each path is detected through the model, the requirement expressions defined for the model are evaluated. The path satisfies any requirement expression that evaluates to TRUE.

Referring to FIG. 8, a user can specify and view requirement expressions via a graphical user interface. The interface shown enables a user to specify each system requirement as a row in a table 222. The table 222 includes columns for a requirement ID 208 and version number 210 for each requirement. This enables a user to quickly correlate requirements with their descriptions in written documents and specify which collections of requirements should be used during path detection (e.g., only version 2 requirements need be satisfied). The requirement ID 208 can also be used as elements in other requirement expressions.

The table also includes columns for a prose description 212 of each requirement and the boolean requirement expression 216. The table can also include a column 214 for specifying a system feature involved in the requirement. A feature may have more than one associated requirement. Additionally, a table column may permit a user to name the row for inclusion in other expressions.

Further, a table can include a "source" column 218 for Hyperlinks (e.g., Universal Resource Locators) which link to external documents describing a requirement.

The information included in the table 222 may be entered manually or imported, for example, from a database, spreadsheet, or a CSV (comma separated value) file. Similarly, the table 222 information may also be exported. Additionally, different requirements may be enabled or disabled by a user.

Referring to FIG. 9, the process can generate a report 224 that describes tests that can be run to test the specified requirements. As shown, the report 224 may be a table that includes a row for each test generated and an indication of the different requirements satisfied by the test. For example, row 231 for test path 3 satisfies requirements 1.0.1 and 1.1.

The report 224 can also summarize test results, for example, by displaying the number of tests satisfying each requirement 226 or displaying the number of requirements a particular path satisfied 232. The report enables a user to understand the completeness of a set of tests, to understand how many of the requirements have been included in the model, to perform optimization, and to detect tests that do not satisfy defined requirements. Based on the report the user can see which paths satisfied the requirement and use the testing programs generated for these paths to test the system being modeled.

The requirements feature described above can also limit (i.e., "filter") the test scripts generated. For

example, a user can specify that test scripts should only be generated for paths satisfying a particular requirement. Thus, only testing programs directed to testing particular features are generated. Additionally requirement information can be output to a CSV (comma separated value) file, spreadsheet, or database.

Referring to FIG. 10, similar to requirements, assertions enable a user to specify an expression for evaluation. However, while a path through a perfectly designed model may not satisfy any requirement expressions, assertions represent expressions that should always be satisfied (e.g., TRUE) when evaluated. Failure to satisfy an assertion can represent significant model flaws needing immediate attention (e.g., when an abnormal or unexpected condition occurs).

A process 240 for determining whether a model complies with a set of assertions includes receiving 242 assertion expressions. A user can specify that an assertion expression be evaluated at different points in the model, for example, before or after entering a particular state, transition, or sub-model. In another embodiment, a designer can specify that an assertion expression should be automatically evaluated before and/or after entering every sub-model element. Additionally, a designer can specify that an assertion expression should be automatically evaluated after each path through the model is detected.

When the process 240 determines 246 a path violates an assertion (i.e., the boolean assertion expression evaluates to FALSE), the process 240 can immediately alert

248 the user of the particular path and other model information that caused the assertion violation. For example, the process 240 can call a model debugger that enables a user to view model information such as the value of different variables, the assertion violated, and model elements in the path that violated an assertion. This enables a user to examine the model context that caused the assertion to fail. The process 240 can further provide an error message and/or provide a display that highlights the path the caused the violation.

Transition Tables:

Referring to FIG. 11, a graphical user interface provides a table 143 model element the user can include in a model. The table 143 can specify multiple sets of data to be included in the generated test.

Referring to FIG. 12, each row can include one or more variable value assignments, for example, each row can include a different value for the PIN model variable 250 and a name of the customer assigned that PIN (not shown). Each row can further include predicate 254 and/or constraint expressions 256. The path detection instructions can select one or more of the rows for each path. Thus, the table 143 provides a convenient mechanism for viewing and defining large sets of data.

In another embodiment, the table also includes columns for specifying a source state and a destination state for each transition row (not shown). This enables an entire model to be displayed as one or more tables of rows.

The tables can be used to automatically generate a graphical display of a model. Similarly, a graphical model could be used to generate corresponding tables. The equivalence of the model and the table enable a user to easily "flip" between the different model representations.

Additionally, the table may offer a column for a name of the row (not shown). The named model element can then be included in other expressions.

Each row of the table 143 can also include a likelihood value 252. The likelihood values can be used to select a row from the table during path detection. Referring also to FIG. 13, a process 258 for selecting a row based on likelihood values includes determining currently eligible rows 260, normalizing the likelihood values of the eligible transitions 262 to produce a probability for each eligible transition, and selecting a transition based on the produced probabilities.

For example, assume the TEST model variable is set to "1" in FIG. 12. Under this assumption, PINs 001, 002, 003, and 004 represent eligible transitions because these transitions satisfy their associated predicate and/or constraint expression(s). As shown, the likelihood values in a table need not add to 1 or 100. For example, adding the likelihood values of the eligible rows (PINs 001, 002, 003, and 004) yields a total of 160. A row (e.g, representing a transition) can be selected by using the total likelihood value and the individual likelihood values of the eligible rows to dynamically create selection ranges for each row. For example, a sample set of ranges may be:

PIN=001	0.000 - 0.062	(e.g., 10/160)
PIN=002	0.063 - 0.188	(e.g., 0.062 + 20/160)
PIN=003	0.189 - 0.750	(e.g., 0.188 + 90/160)
PIN=004	0.751 - 0.999	(e.g., 0.750 + 40/160).

Thereafter, a row can be selected by generating a random number between 0 and 1 and selecting the transition having a range covering the generated number. For example, a random number of 0.232 would result in the selection of the transition setting the PIN variable to "003". Use of probabilities enables a model to be tested using data that reflects actual usage. Additionally, the use of probabilities enables a small set of rows to represent a large set of data. Further, normalizing likelihood values to produce probabilities enables the path detection instructions to process probabilities with different combinations of eligible rows.

Other embodiments can include variations of the features describe above. For example, probabilities and/or likelihood values can be assigned to transitions with or without the use of table model elements. Additionally, though the determination of eligible transitions and normalizing their respective likelihood values provides a designer with flexibility, these actions are not required to take advantage of the benefits of including probabilities in the model.

<u>Importing Data into the Model:</u>

The rows in the table and other model information can be hand entered by a user. Alternatively, a user can import the data from an external source. Referring to FIG. 14, a process 250 enables users to import data into a model by specifying 252 an external information source for importing 254 into the model. For example, referring to FIG. 15, for, a user can specify a file name of a CSV (Comma Separated Value) file. The first line 266 of the CSV file defines table schema information such as the table variables and their corresponding data types. For example, as shown the variable named PIN has been type-cast as a number 268. Subsequent information in the CSV is paired with the variables defined in the first line 266. For example, the number 001 is paired with the variable PIN while the string "FirstPIN" is paired with the string variable named OtherInformation.

A database or spreadsheet could also be used as a source of external data. For example, a user could specify a relational database view or table. In response, instructions can automatically access the database to obtain schema information for the table. For example, an SQL (Structured Query Language) select command can be used to determine the variables and data included in a particular table or view and output this information to a CSV file. For interfacing with different types of data sources, the instructions may support ODBC (Open Database Connectivity) drivers.

Importing data from an external source can relieve a user from having to define a large number of transitions

between states by hand. However, the importing capability is not limited to transitions. Additionally, the imported data can reflect actual testing conditions. For example, a log file produced by a system in actual use can provide a valuable source of data for the model.

Specifying a Mix of Paths:

Referring to FIG. 16, a process 300 enables a user to control the mix of paths outputted during path detection. The process 300 enables a user to specify 302 a desired mix of generated tests. For example, a user can specify a percentage (or ratio) of paths that include a particular model element or that satisfy a particular expression. During path detection, instructions track the current mix of paths (e.g., how many paths are in the mix and how many paths include the model element) and determine 306 whether a newly detected path brings the mix closer to the user specified percentage(s). If so, the newly detected path is saved in the mix. Otherwise, the path is discarded.

Many different procedures for determining whether a detected path brings the mix close to the user specified percentages could be used. For example, one procedure saves a detected path if the path satisfies any specified expression that is currently under-represented in the mix generated thus far. For example, referring to FIG. 17, a bank machine model 320 includes states 322-330 that represent different bank machine transactions. As shown, a user has specified that the mix of paths generated should include 40% 332 withdrawals 322 and 35% 334 checking-to-

savings 330 transfers. Assume that after nine paths, two paths have included withdrawals 332 (i.e., 22%) and three have included checking-to-savings 330 (i.e., 33%) transactions. Further assume a newly generated path included the model elements A - T_{AB} - B - T_{BF} - F. This path includes a withdrawal 332, but no checking-to-savings 330 transactions. Since the running percentage of withdrawals 332 is only 22% as compared to a target of 40%, the new path will be included in the mix.

Other embodiments use different techniques for determining whether a path improves the mix of tests. For example, in the previous example, including the new path improved the percentage of withdrawals 332 from 22% to 33%, but would lower the percentage of checking-to-savings 330 transactions to 30%. Thus, saving the new path in the mix would bring the percentage of withdrawals 332 in the mix closer to the withdrawal target by 8% while bringing the percentage of checking-to-savings 330 by 3% away from its target. One embodiment totals the amount each current percentage is from its target percentage and compares this to the same total if the current path were saved in the mix. If the total would be reduced by inclusion of the path, the path is saved in the mix. Additionally, in some embodiments, a user can specify that some target percentages take priority over others.

The specified targets need not add up to 100% as each test mix expression is independent of the other expressions. For example, as shown in FIG. 17, the targets only totalled 75%. This gives a user flexibility in using

the test mix feature.

By specifying a mix of paths, a user can generate tests for model features of interest without defining additional expressions to control model behavior. Additionally, the technique enables a user to produce tests relevant to areas of interest or that mimic behavior of interest.

Other Embodiments:

The techniques described here are not limited to any particular hardware or software configuration; they may find applicability in any computing or processing environment. The techniques may be implemented in hardware or software, or a combination of the two. Preferably, the techniques are implemented in computer programs executing on programmable computers that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code is applied to data entered using the input device to perform the functions described and to generate output information. The output information is applied to one or more output devices.

Each program is preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program is preferable stored on a storage medium or device (e.g., CD-ROM, embedded ROM, hard disk or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described in this document. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other embodiments are within the spirit and scope of the appended claims.

What is claimed is:

1. A method of using a computer to analyze an extended finite state machine model of a system, the model having states interconnected by transitions, the method comprising:

receiving at least one requirement expression;

determining at least one path of states and transitions through the model;

evaluating at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression; and

generating a report based on the evaluating.

2. The method of claim 1, wherein the report comprises a report of paths satisfying at least one requirement.

3. The method of claim 1, wherein the report comprises a report of requirements satisfied by a path.

4. The method of claim 1, wherein the requirement expression comprises a boolean expression.

5. The method of claim 1, wherein the requirement expression comprises at least one of the following: a variable, an operator, a state, a transition, a sub-model, a table-model, and another requirement expression.

6. The method of claim 1, further comprising generating testing programs only for paths satisfying a specified requirement or set of requirements.

7. A method of using a computer to analyze an extended finite state machine model of a system, the model having states interconnected by transitions, the method comprising:

receiving a requirement expression;

determining at least one path of states and transitions through the model;

evaluating at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression; and

including the requirement expression as an element in a different expression, the evaluation of the requirement expression in the different expression depending on the determination of whether the path satisfies the requirement expression.

8. A method of using a computer to analyze an extended finite state machine model of a system, the model having states interconnected by transitions, the method comprising:

receiving at least one assertion expression;

evaluating the assertion expression based on a path being determined through the model; and

if the evaluation indicates the path being determined fails to satisfy the assertion expression,

indicating the failure.

9. The method of claim 8, further comprising halting model path determination.

10. The method of claim 8, wherein the assertion expression comprises a boolean expression.

11. The method of claim 8, wherein the assertion expression comprises at least one of the following: a variable, an operator, a state, a transition, a sub-model, a table-model, and a requirement.

12. The method of claim 8, further comprising initiating a debugger if the evaluation indicates the path being determined fails to satisfy the assertion expression.

13. The method of claim 8, further comprising receiving user input specifying when the assertion expression is evaluated.

14. The method of claim 13, wherein specifying when the assertion expression is evaluated comprises specifying a model element.

15. The method of claim 8, further comprising automatically evaluating the assertion expression after a complete path through the model has been determined.

16. The method of claim 8, further comprising

automatically evaluating the assertion expression as model elements are added to a path being determined.

17. A computer program product, disposed on a computer readable medium, for analyzing an extended finite state machine model of a system, the model having states interconnected by transitions, the computer program product including instructions for causing a processor to:

receive at least one requirement expression;

determine at least one path of states and transitions through the model;

evaluate at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression; and

generate a report based on the evaluating.

18. A computer program product, disposed on a computer readable medium, for analyzing an extended finite state machine model of a system, the model having states interconnected by transitions, the computer program product including instructions for causing a processor to:

receive a requirement expression;

determine at least one path of states and transitions through the model;

evaluate at least one of the requirement expressions based on at least one of the determined paths through the model to determine whether the path satisfies the requirement expression; and

include the requirement expression as an element in

a different expression, the evaluation of the requirement expression in the different expression depending on the determination of whether the path satisfies the requirement expression.

19. A computer program product, disposed on a computer readable medium, for analyzing an extended finite state machine model of a system, the model having states interconnected by transitions, the computer program product including instructions for causing a processor to:

receive at least one assertion expression;

evaluate the assertion expression based on a path being determined through the model; and

if the evaluation indicates the path being determined fails to satisfy the assertion expression, indicate the failure.

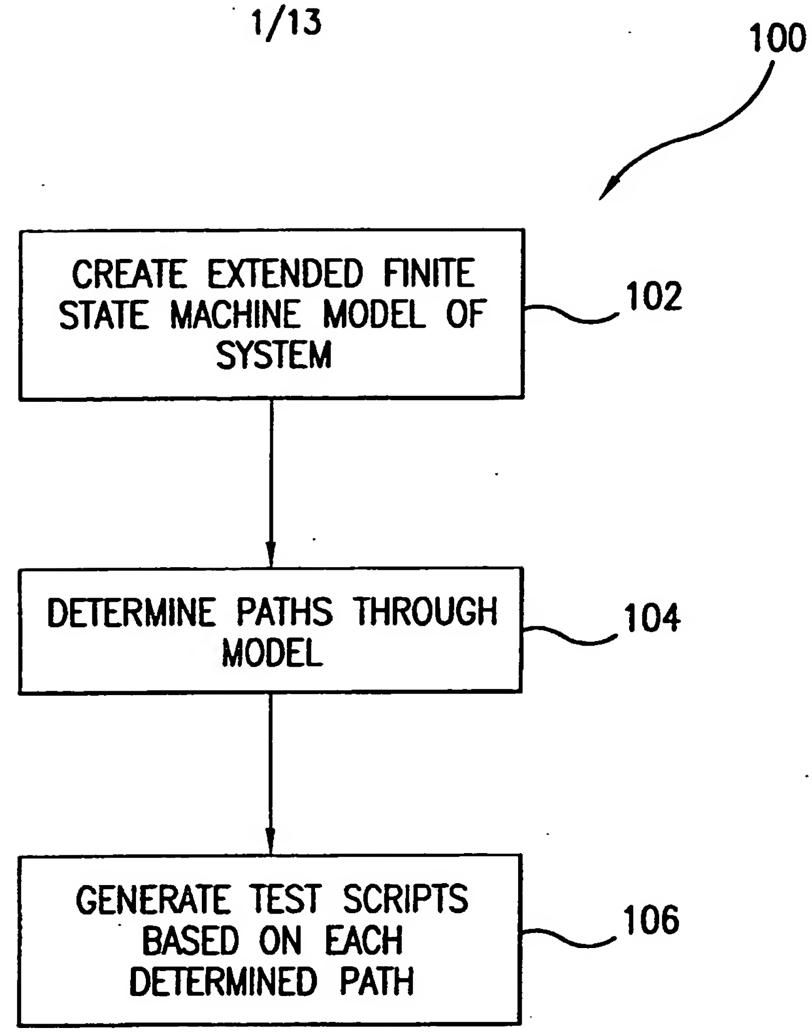


FIG.1
PRIOR ART

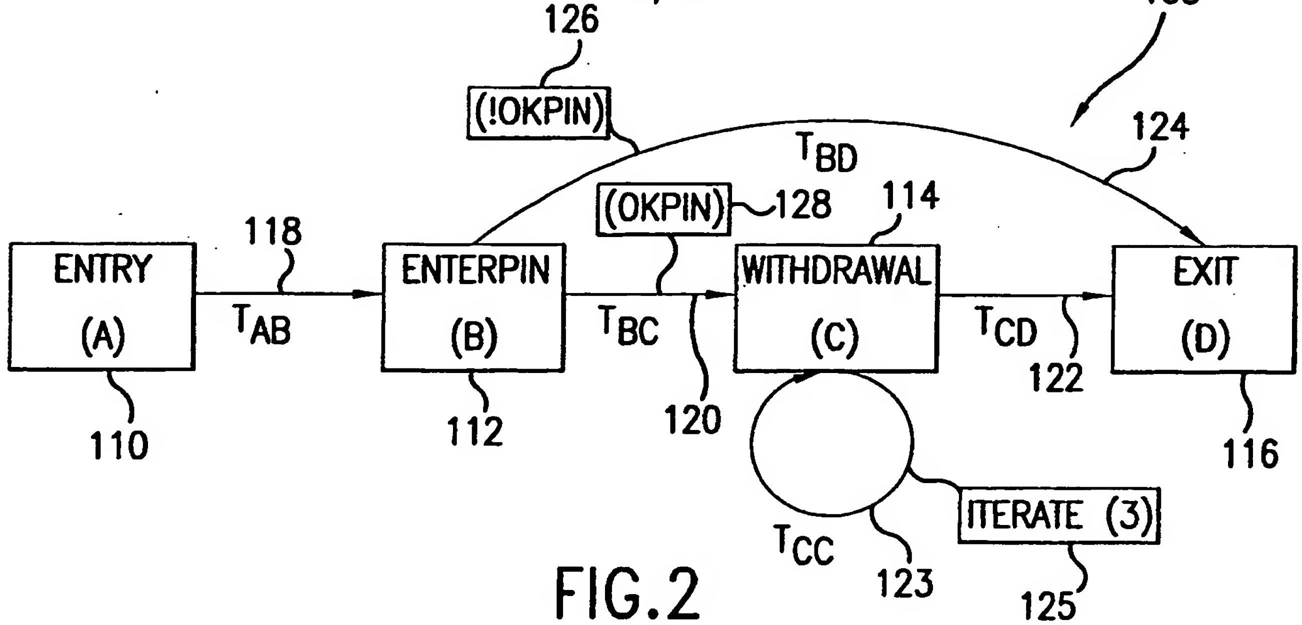


FIG.2

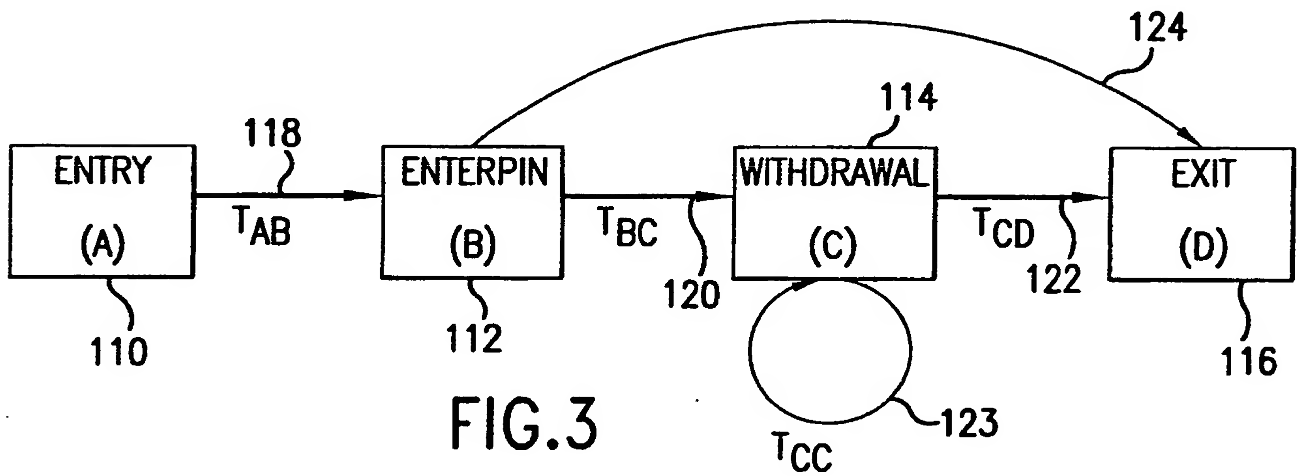


FIG.3

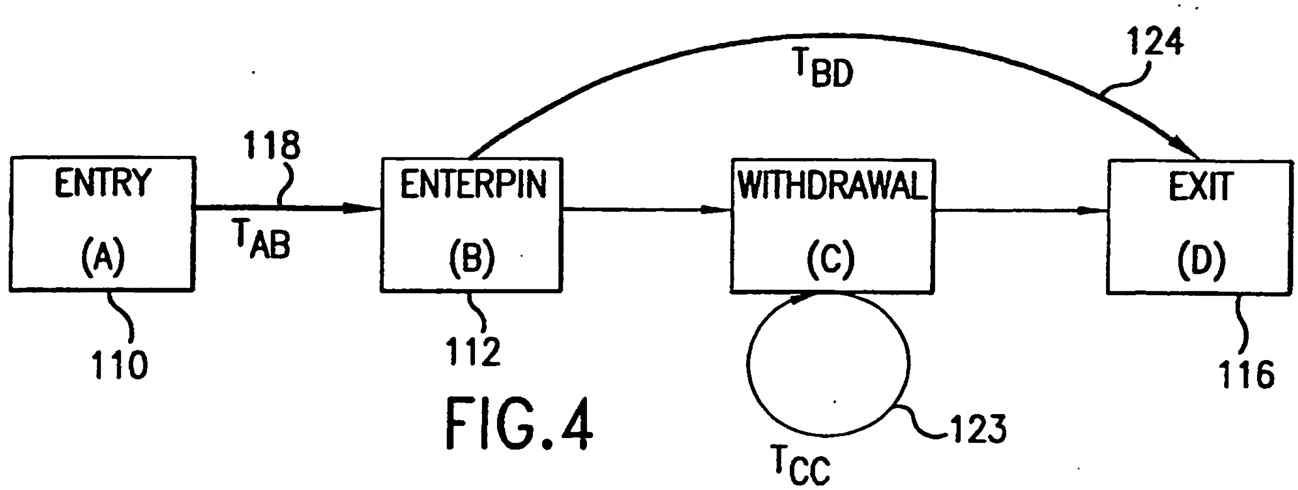


FIG.4

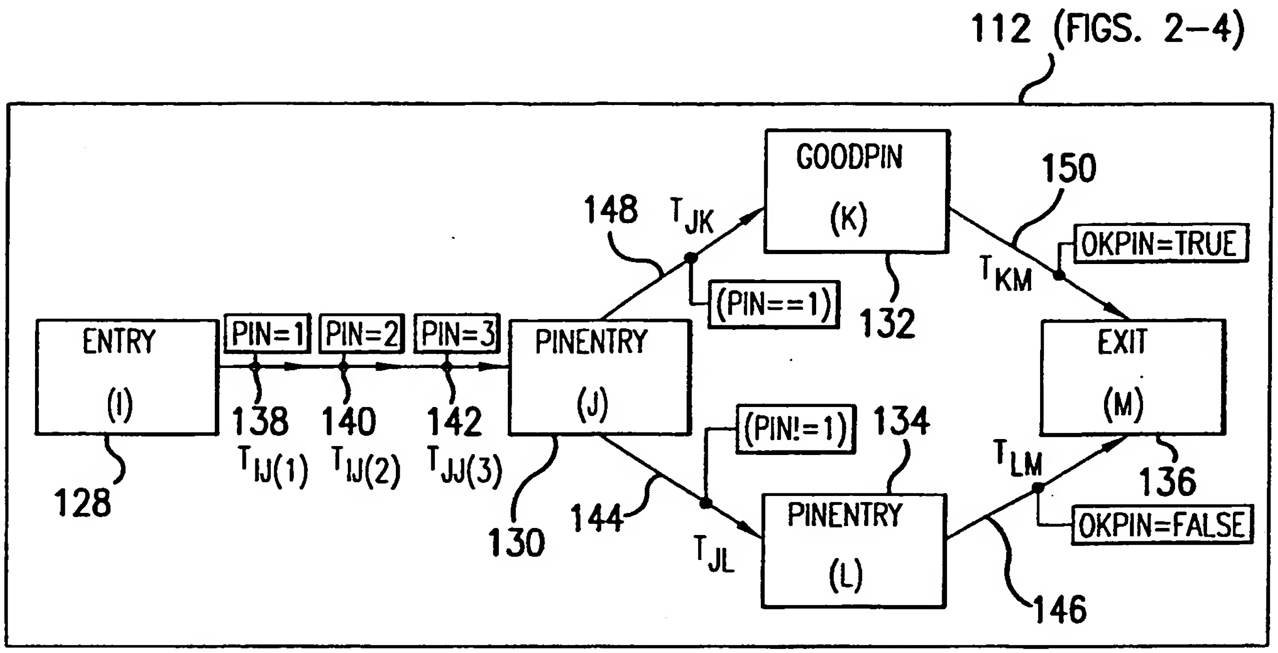

112 (FIGS. 2-4)
GOODPIN
(K)
150
148
T_{JK}
OKPIN=TRUE
T_{KM}
(PIN==1)
132
ENTRY
(I)
PIN=1
PIN=2
PIN=3
PINENTRY
(J)
EXIT
(M)
138
140
142
$T_{IJ}(1)$
$T_{IJ}(2)$
$T_{JJ}(3)$
(PIN!=1)
134
T_{LM}
136
128
130
144
T_{JL}
PINENTRY
(L)
OKPIN=FALSE
146

FIG.5

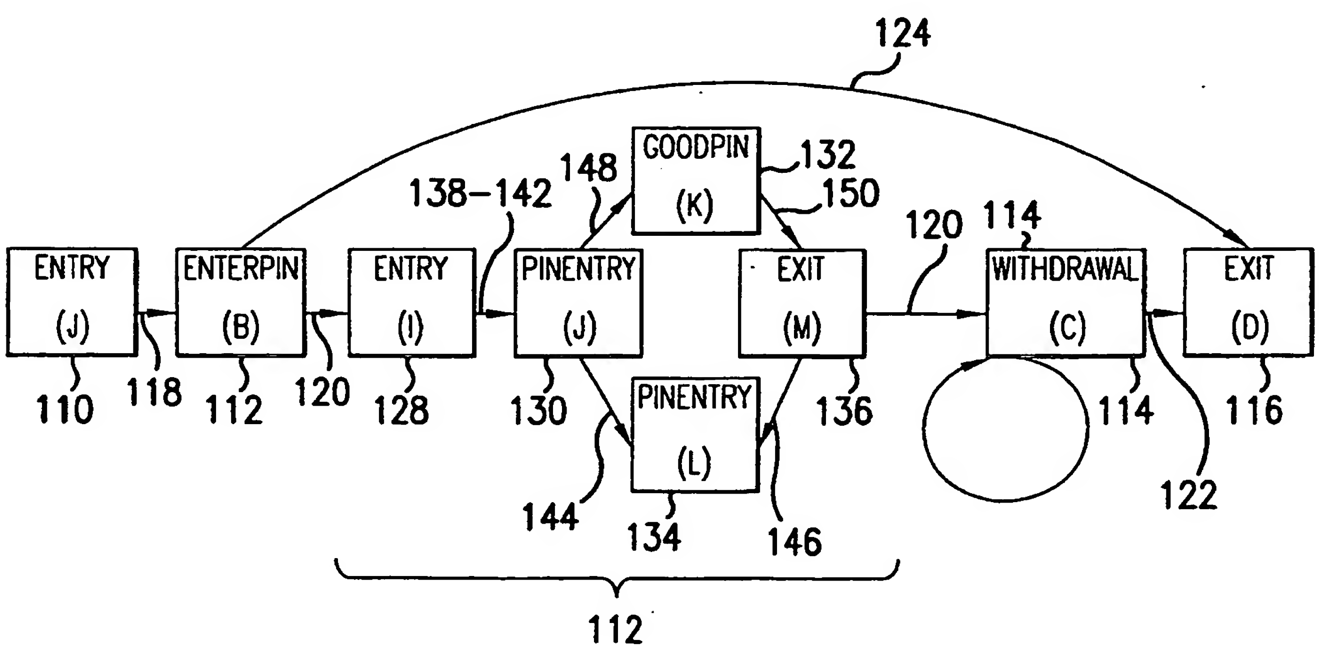

124
GOODPIN
(K)
132
150
148
138-142
120
114
ENTRY
(J)
ENTERPIN
(B)
ENTRY
(I)
PINENTRY
(J)
EXIT
(M)
WITHDRAWAL
(C)
EXIT
(D)
110
118
112
120
128
130
136
114
116
PINENTRY
(L)
122
144
134
146
112

FIG.6

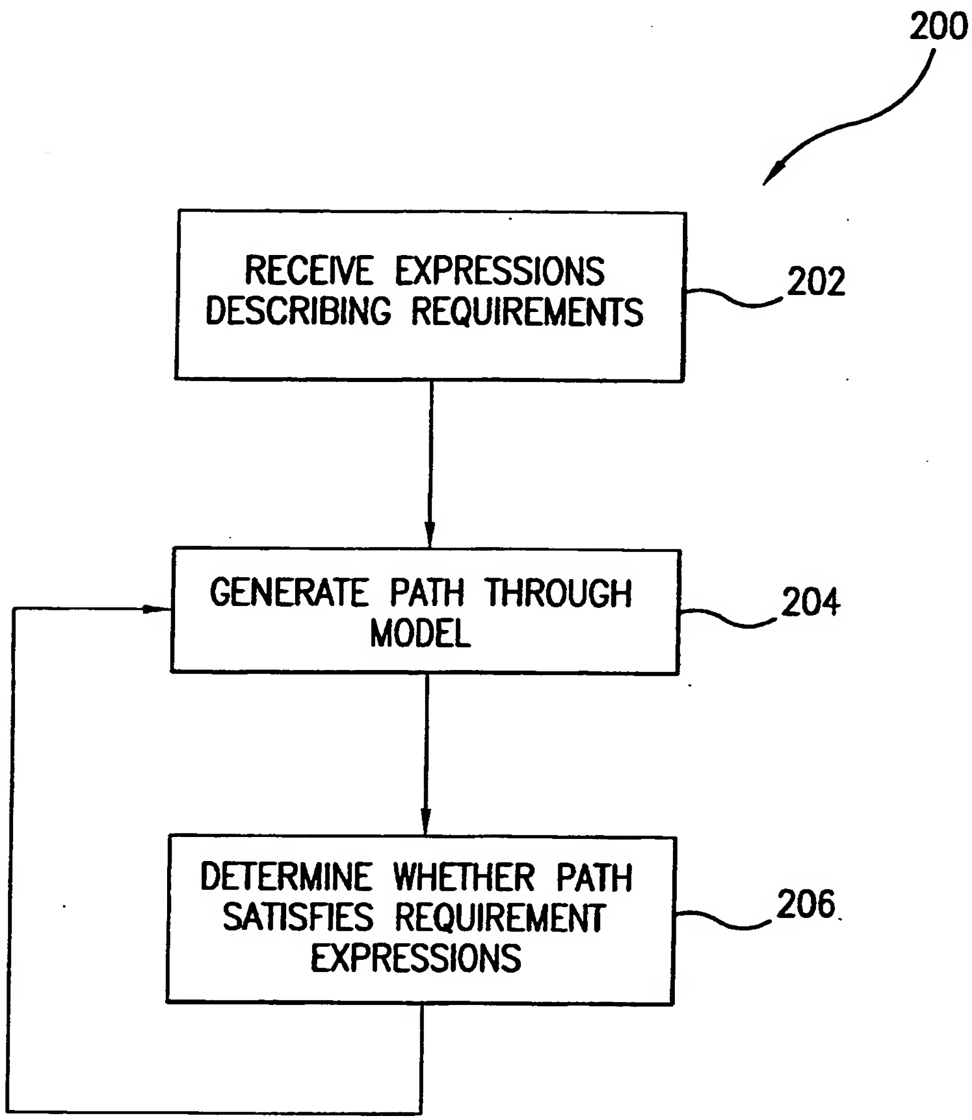

200
RECEIVE EXPRESSIONS DESCRIBING REQUIREMENTS
202
GENERATE PATH THROUGH MODEL
204
DETERMINE WHETHER PATH SATISFIES REQUIREMENT EXPRESSIONS
206

FIG.7

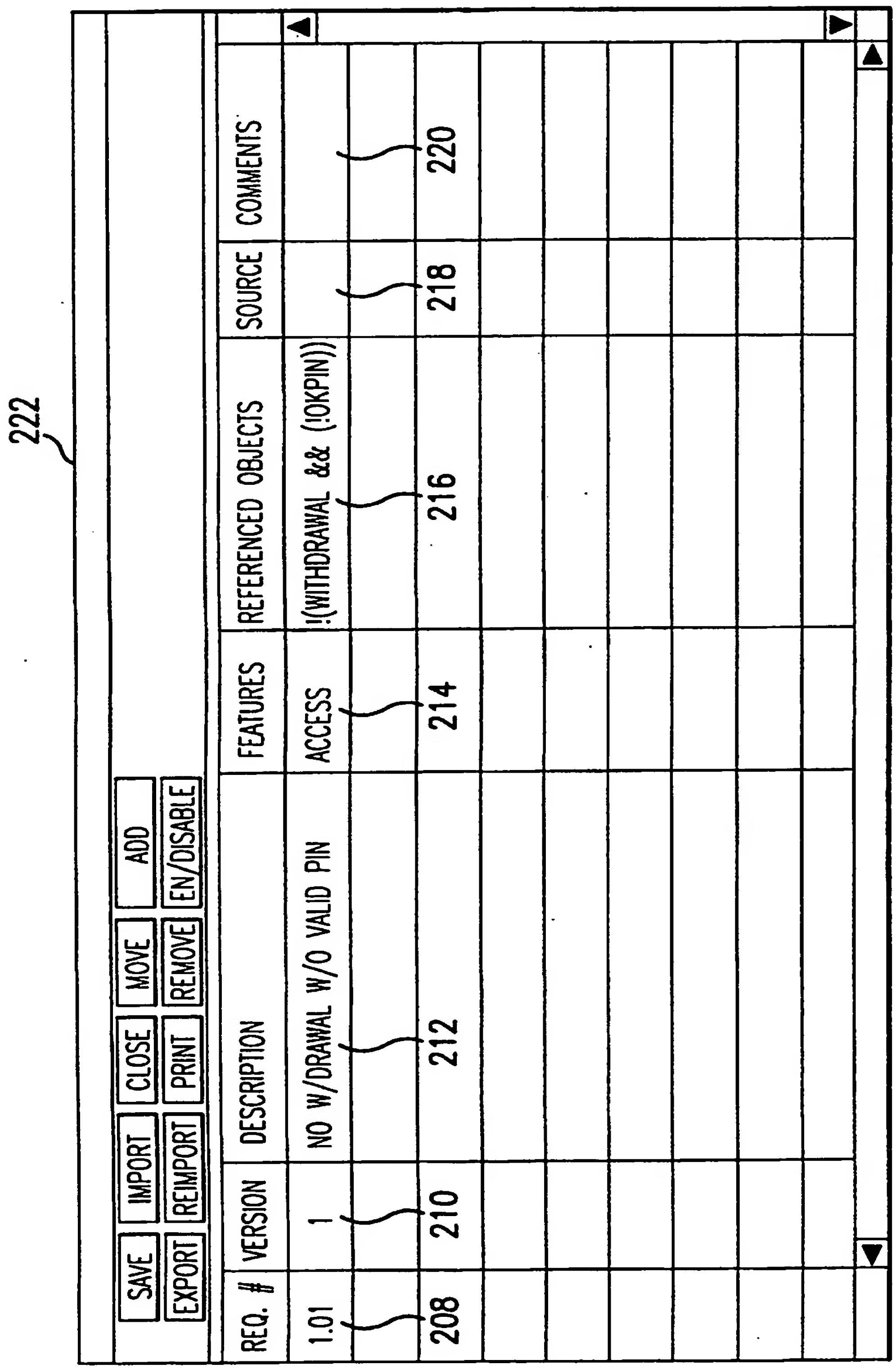

222
SAVE
IMPORT
CLOSE
MOVE
ADD
EXPORT
REIMPORT
PRINT
REMOVE
EN/DISABLE
REQ. #
VERSION
DESCRIPTION
FEATURES
REFERENCED OBJECTS
SOURCE
COMMENTS
1.01
1
NO W/DRAWAL W/O VALID PIN
ACCESS
!(WITHDRAWAL && (!OKPIN))
208
210
212
214
216
218
220

FIG.8

		REQUIREMENT NUMBERS						
	TOTAL TEST PER REQ.	2	3	2	3	3	0	1
TOTAL REQ PER TEST	TEST NUMBER	1.0	1.0.1	1.0.2	1.1	1.1.1	1.2	1.2.1
3	1		X			X		X
1	2	X						
2	3		X		X			
1	4				X			
0	5							
2	6		X		X			
3	7	X		X		X		
0	8							
1	9					X		
1	10			X				

224 226 228 231 230 232

FIG.9

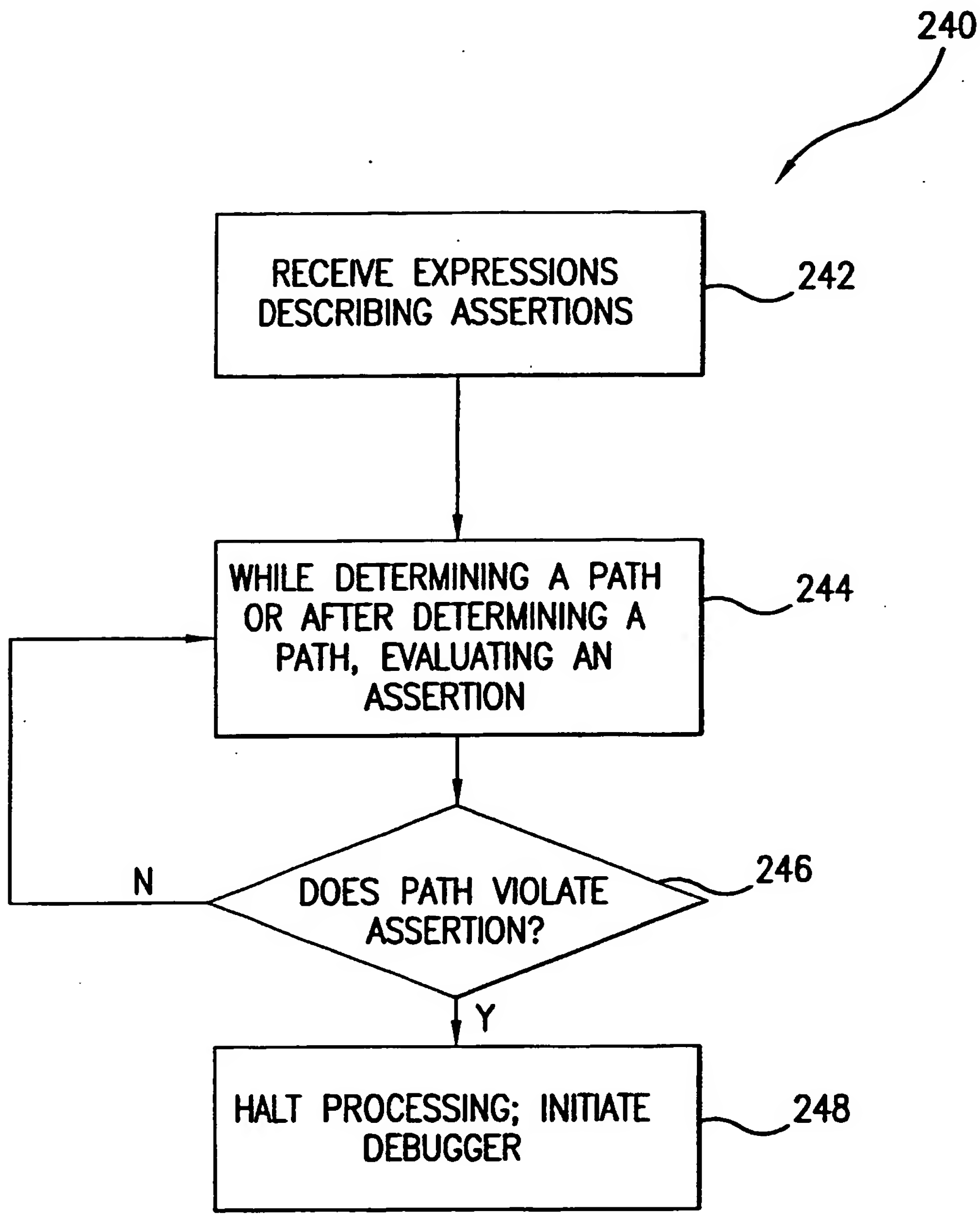

240
RECEIVE EXPRESSIONS DESCRIBING ASSERTIONS
242
WHILE DETERMINING A PATH OR AFTER DETERMINING A PATH, EVALUATING AN ASSERTION
244
DOES PATH VIOLATE ASSERTION?
246
N
Y
HALT PROCESSING; INITIATE DEBUGGER
248

FIG.10

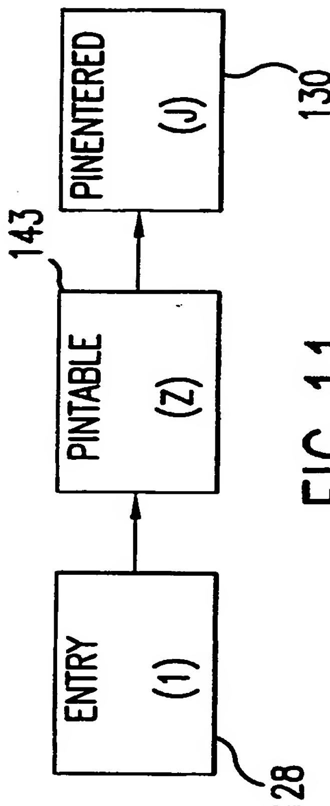

ENTRY
(1)
128
PINTABLE
(Z)
143
PINENTERED
(J)
130

FIG. 11

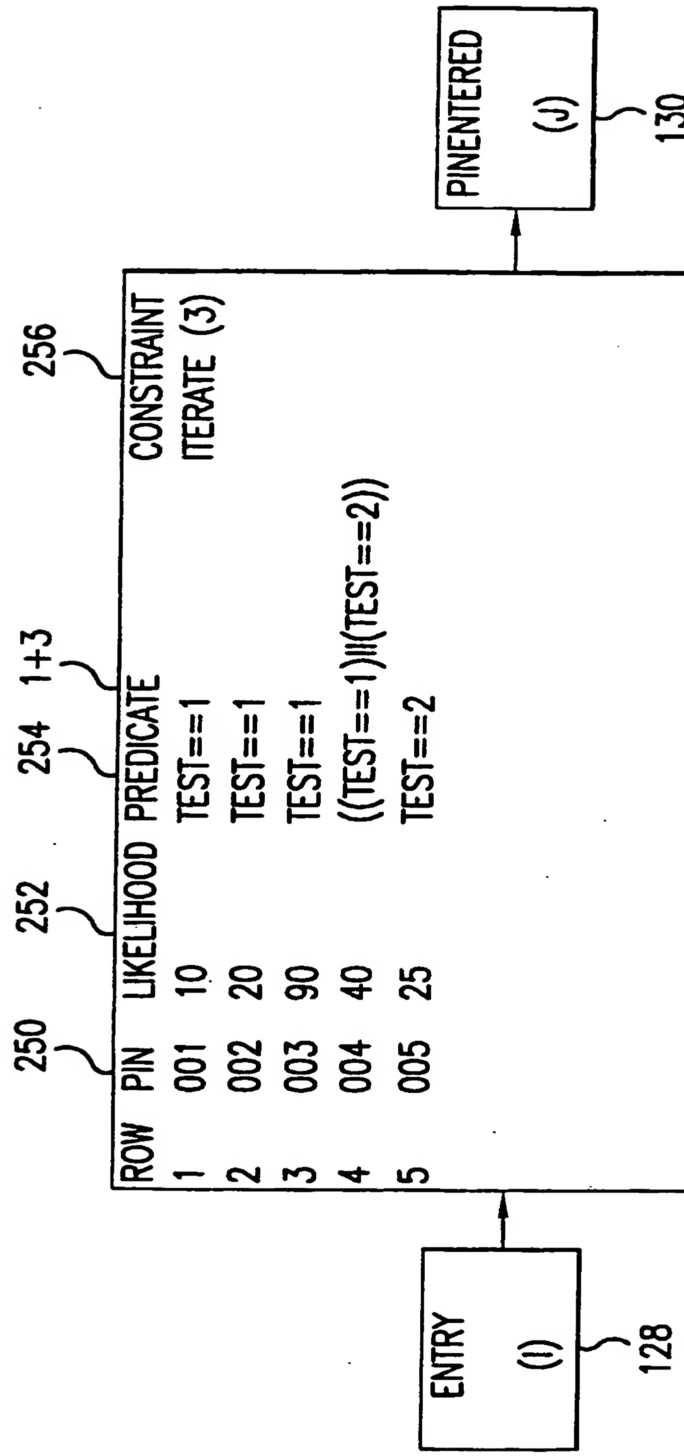

ENTRY
(I)
128
250
252
254
1+3
256
ROW PIN LIKELIHOOD PREDICATE CONSTRAINT
ITERATE (3)
1 001 10 TEST==1
2 002 20 TEST==1
3 003 90 TEST==1
4 004 40 ((TEST==1)||(TEST==2))
5 005 25 TEST==2
PINENTERED
(J)
130

FIG. 12

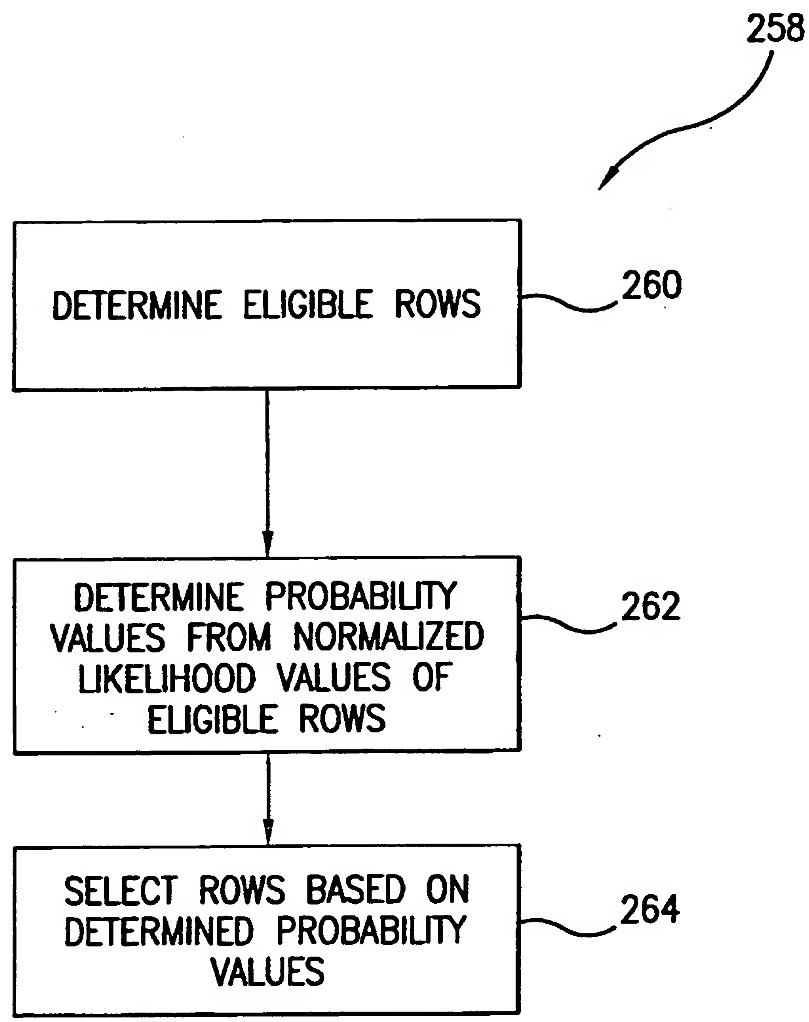

258
DETERMINE ELIGIBLE ROWS
260
DETERMINE PROBABILITY VALUES FROM NORMALIZED LIKELIHOOD VALUES OF ELIGIBLE ROWS
262
SELECT ROWS BASED ON DETERMINED PROBABILITY VALUES
264

FIG.13

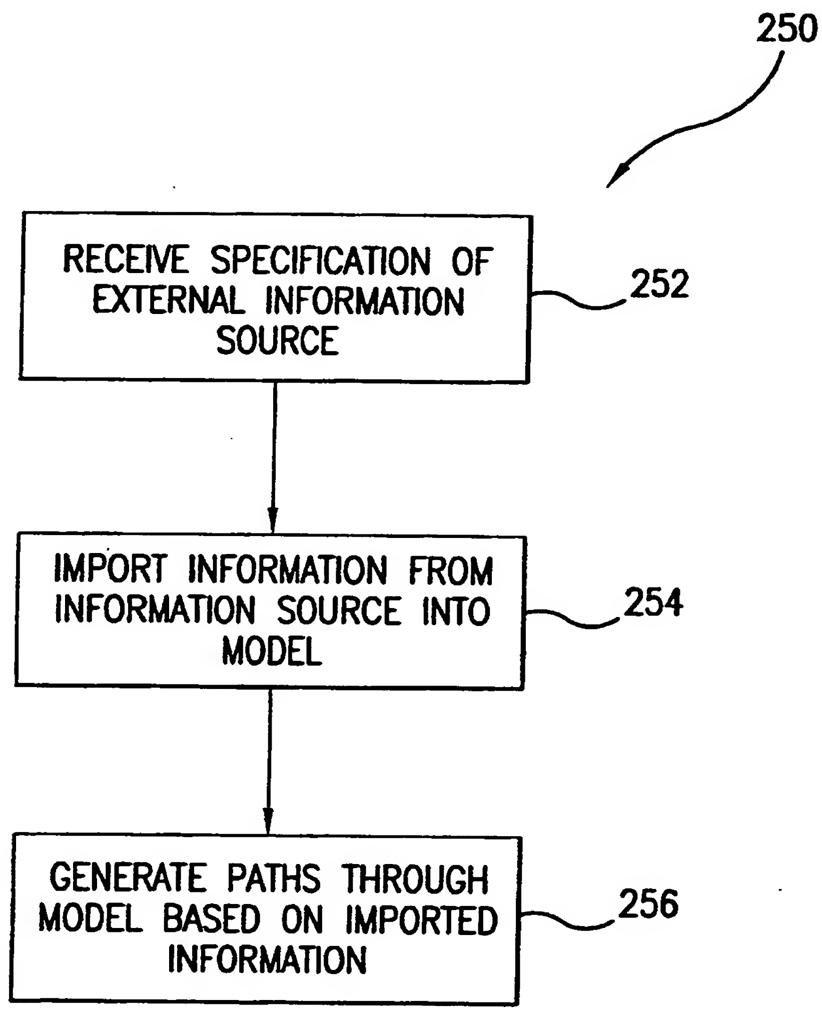

250
RECEIVE SPECIFICATION OF EXTERNAL INFORMATION SOURCE
252
IMPORT INFORMATION FROM INFORMATION SOURCE INTO MODEL
254
GENERATE PATHS THROUGH MODEL BASED ON IMPORTED INFORMATION
256

FIG.14

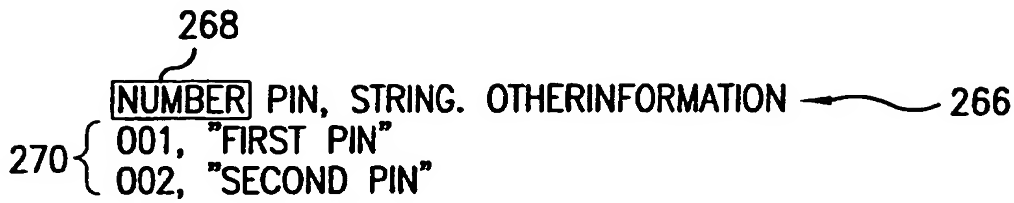

268
NUMBER PIN, STRING. OTHERINFORMATION
266
270
001, "FIRST PIN"
002, "SECOND PIN"

FIG. 15

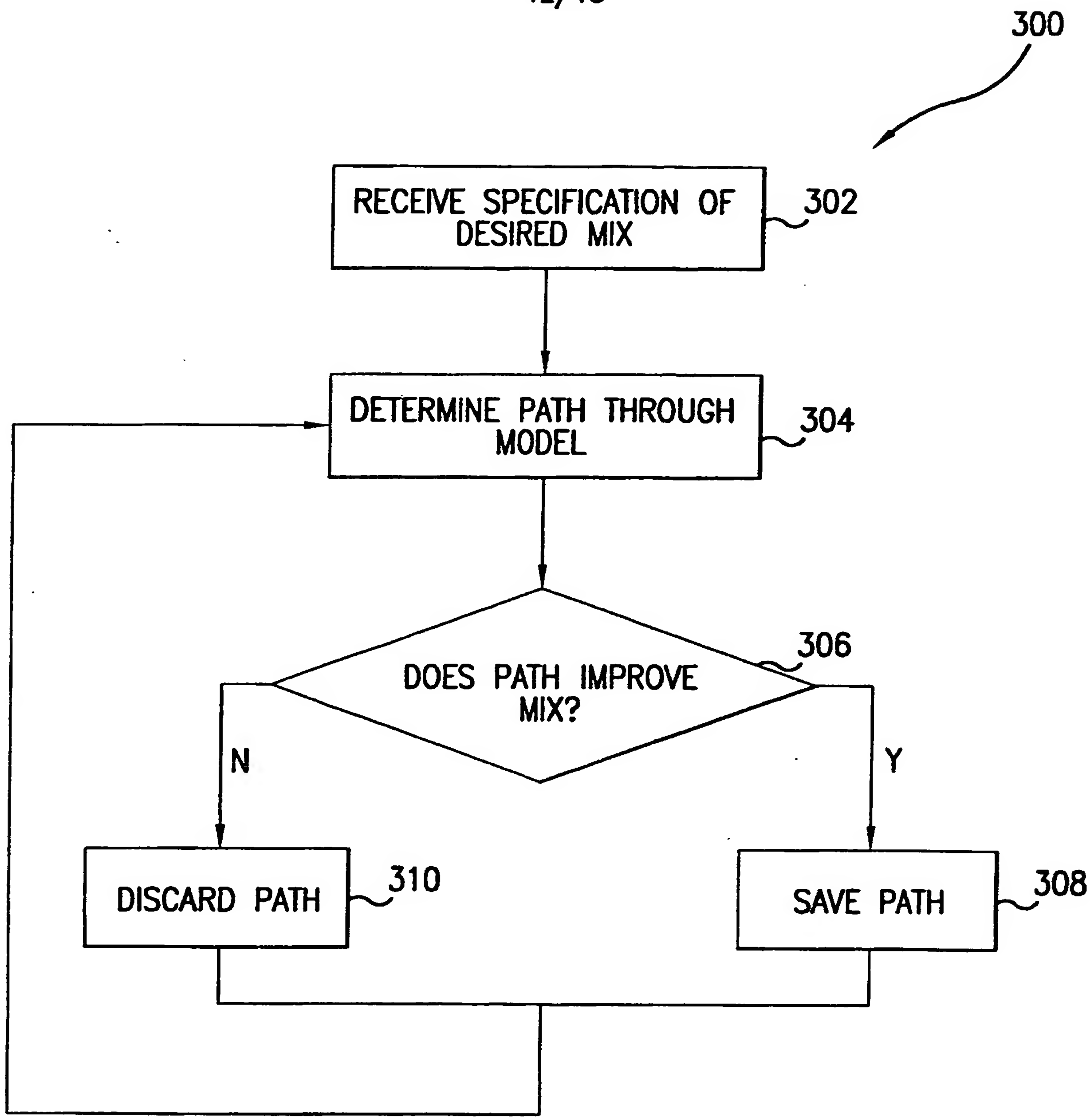

300
RECEIVE SPECIFICATION OF DESIRED MIX
302
DETERMINE PATH THROUGH MODEL
304
DOES PATH IMPROVE MIX?
306
N
Y
DISCARD PATH
310
SAVE PATH
308

FIG. 16

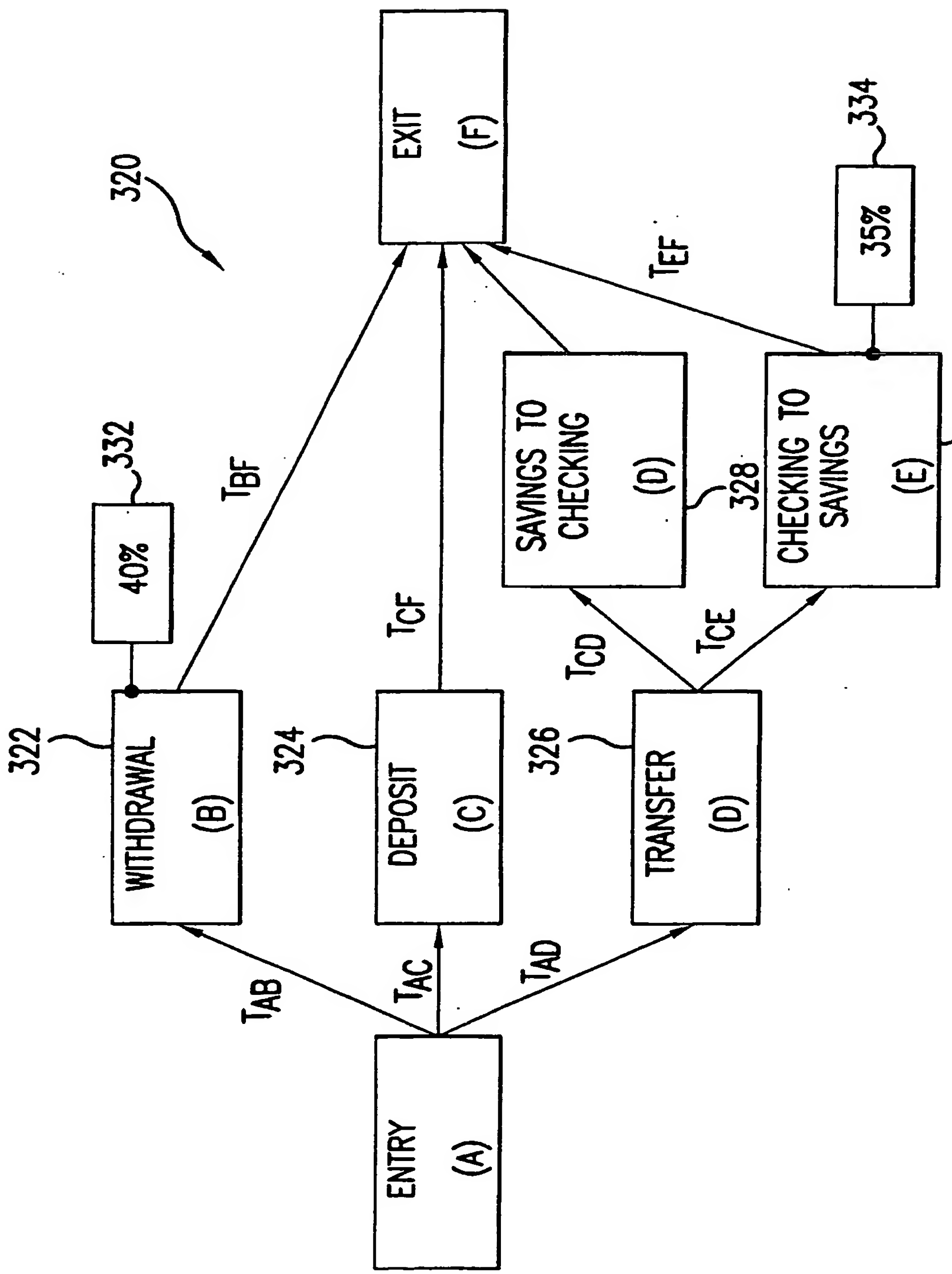

320
ENTRY
(A)
T_{AB}
T_{AC}
T_{AD}
322
WITHDRAWAL
(B)
324
DEPOSIT
(C)
326
TRANSFER
(D)
332
40%
T_{BF}
T_{CF}
T_{CD}
T_{CE}
SAVINGS TO CHECKING
(D)
328
CHECKING TO SAVINGS
(E)
330
334
35%
T_{EF}
EXIT
(F)

FIG. 17

INTERNATIONAL SEARCH REPORT

Int ...tional Application No
PCT/US 00/14248

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, INSPEC, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	S. MEYER, L. APFELBAUM: "use-cases are not requirements" TESTMASTER PRODUCT LITERATURE. TECHNICAL PAPER., 'Online! 19 March 1999 (1999-03-19), pages 1-12, XP002149951 Retrieved from the Internet: <URL:http://www.teradyne.com/prods/sst/Literature/literature.html> 'retrieved on 2000-10-10!	1-6,8, 10,11, 13-17,19
Y	page 8, paragraph 6.1 -page 11, paragraph 7	7,9,12, 18

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
20 October 2000	07/11/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Renault, S

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/14248

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5 394 347 A (TREMBLAY SYLVIA C ET AL) 28 February 1995 (1995-02-28) column 5, line 40 - line 47 column 6, line 16 - line 18 column 16, line 11 - line 17	7,9,18
Y	L. APFELBAUM: "Spec-based tests make sure telecom software works" IEEE SPECTRUM, vol. 34, no. 11, 11 November 1997 (1997-11-11), pages 77-83, XP002149691 page 81, column 1, line 6 -page 82, column 1, line 19	12
A	L. APFELBAUM: "Automated Functional Test Generation" AUTOTESTCON'95. SYSTEMS READYNESS: TEST GENERATION FOR THE 21ST CENTURY. CONFERENCE RECORD, 1995, pages 101-107, XP002149952 1995 IEEE the whole document	1-19

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 00/14248

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5394347 A	28-02-1995	NONE	